SYS863
Internet industriel des objets

Laboratoire 2

Lokman Sboui, Ph.D.
Professeur en génie des systèmes
École de technologie supérieure
Montréal, Québec

Hiver 2023

Table des Matières - Laboratoire 2

1. Introduction

1.1 Objectifs

Le 2e laboratoire se veut une extension du laboratoire #1 et une introduction à la couche réseau IIoT. Après l'exploration locale des données, on abordera le protocole MQTT. On communiquera aussi avec des capteurs montés en Bluetooth via un réseau LoRaWAN. La majorité de vos travaux seront sur la plateforme Node-RED que vous allez maitriser d'ici la fin de ce laboratoire.

Dans le laboratoire, on considère que le contrôleur de température est connecté. Les gains du PID sont fixés une fois pour toute dans le programme (vous ne pouvez pas modifier les gains d'un thermostat intelligent acheté à la quincaillerie). Il faudra avoir accès à la température mesurée et pouvoir envoyer la consigne à distance via le WiFi.

1.2 Évaluation

Ce laboratoire représente 10% de la note finale. Un rapport, le listing final et le fichier Node-RED doivent être remis via le site Moodle du cours. Les livrables demandés seront identifiés avec ce symbole "↬" et seront écrit en rouge.

1.3 Format à utiliser

Le rapport doit être soumis sous forme de fichier .pdf avec le nom **EquipeX_Lab2_Rapport.pdf** avec X la lettre de votre équipe. Pour les autres fichiers (ex :listings en C, et .json de Node-RED) le nom doit être : **EquipeX_Lab2.c** et **EquipeX_Lab2.json**. Pour les fichiers .csv le nom est indiqué dans les instructions.

Le rapport de projet de session doit se conformer le plus possible au format suivant :

- ▷ Page de présentation ;
- ▷ Buts et objectifs du projet de session ;

- ▷ Expliquer votre démarche de conception ;
- ▷ Expliquer ce que vous observez ;
- ▷ Pour chaque programme que l'on vous demande d'annexer au rapport de projet de session :
  - ◎ Explication du fonctionnement de chaque programme (ce peut être via un ordinogramme) ;
  - ◎ " Listing " des fichiers sources (les sections ou fonctions modifiées ou ajoutées) ;
- ▷ Problématiques rencontrées ;
- ▷ Conclusion.

2. Travaux du Laboratoire 2

2.1 Partie 1 - Sauvegarde locale des données

2.1.1 Les fichiers CSV

Dans cette partie, nous allons sauvegarder les données de la lampe 12V et le thermocouple dans un fichier CSV localement dans la mémoire du Raspberry Pi. Les fichiers CSV (Comma Separated Values) sont un format de fichier de données simple qui utilise des virgules pour séparer les valeurs dans un fichier. Chaque ligne représente une entrée de données et chaque colonne représente une valeur différente pour cette entrée. Les fichiers CSV peuvent être ouverts et modifiés avec des logiciels tels que Microsoft Excel, Google Sheets ou d'autres tableurs. Ils sont souvent utilisés pour stocker et partager des données, car ils peuvent être lus par de nombreuses applications et sont faciles à utiliser.

2.1.2 Créer un fichier CSV en C

Avant de commencer à sauvegarder le fichier CSV, nous devons effectuer quelques préparations. Tout d'abord, nous devons définir le nom du fichier à enregistrer. Nous allons utiliser la variable `FILE *fp` pour stocker le fichier. Nous devons également définir les colonnes du fichier CSV et leurs en-têtes respectifs. Une fois les préparations terminées, nous pouvons maintenant ouvrir le fichier CSV à l'aide de `fopen()`. Ensuite, nous pouvons écrire les données dans le fichier CSV à l'aide de fprintf() : `fprintf(fp, "Date␣et␣heure,Adresse␣MAC,Emplacement,Temperature\n")`. Enfin, nous pouvons fermer le fichier avec `fclose()`.

Voici, ci-dessous, un exemple de programme en C pour enregistrer les données dans un fichier CSV. Ce code utilise les bibliothèques `stdio.h` et `time.h` pour capturer la date et l'heure actuelles et les enregistrer dans un fichier CSV. Le fichier est créé en utilisant la fonction `fopen()` et les données sont écrites dans le fichier en utilisant la fonction `fprintf()`. La fonction `strftime()` est utilisée pour formater la date et l'heure en un format de chaîne spécifique pour être enregistré dans le fichier CSV. Les données enregistrées incluent la date et l'heure actuelles, l'adresse MAC, l'emplacement et la

température. Finalement, le fichier est fermé en utilisant la fonction `fclose()`. Le résultat semble au tableau 2.1.

Listing 2.1 – Créer et sauvegarder un fichier CSV contenant l'heure et la date actuelle

```c
/*
 * SauvegardeCSV.c
 * Exemple de programme pour créer et sauvegarder
 * un fichier CSV contenant l'heure et la date actuelle
 */

#include <stdio.h>
#include <time.h>

char date_et_heure[20];

int main() {

// Capturer l'heure et la date actuelle
time_t t = time(NULL);
struct tm *tm = localtime(&t);
strftime(date_et_heure, 20, "%Y-%m-%d %H:%M:%S",tm);

// Écrire les données dans un fichier CSV

FILE *fp = fopen("temperatures.csv", "w");
fprintf(fp, "Timestamp,Adresse MAC,Emplacement,Temperature\n");
fprintf(fp, "%s,%s,%s,%.1f\n", date_et_heure, "11:22:33:44:55:66", "Poste 13
    ", 24.5);

fclose(fp);

  return 0;
}
```

	A	B	C	D
1	Timestamp	Adresse MAC	Emplacement	Température
2	2022-12-18 01:28:20	b8:27:eb:d4:4c:a0	Poste 13	22.25
3	2022-12-18 01:28:25	b8:27:eb:d4:4c:a0	Poste 13	22.50
4	2022-12-18 01:28:30	b8:27:eb:d4:4c:a0	Poste 13	22.75
5	2022-12-18 01:28:35	b8:27:eb:d4:4c:a0	Poste 13	22.50
6	2022-12-18 01:28:40	b8:27:eb:d4:4c:a0	Poste 13	22.75
7				
8				
9				
10				
11				

FIGURE 2.1 – Exemple du fichier .csv

2.1.3 Trouver l'équation de la variation des données en Excel

On s'interesse à trouver l'équation d'un tableau avec Microsoft Excel et la courbe de tendance afin de carcatereiser notre systeme (Lampe+thermocouple). Pour ce faire, suivez les étapes suivantes :

- Ouvrez le fichier CSV Excel et entrez vos données dans un tableau. **Attention : En Excel (version française) les décimales sont séparées par des virgules au lieu des points !**
- Sélectionnez les deux colonnes qui vont donner la courbe de varation (Timestamp et Température)
- Allez dans l'onglet "Insertion" et cliquez sur "Graphique". Sélectionnez "Nuages de points".
- Une fois que vous avez créé votre graphique, cliquez droit sur la courbe de tendance et sélectionnez "Ajouter une courbe de tendance".
- Dans le menu "Format de courbe de tendance", allez à l'onglet "Options de courbe" et choisissez le type de courbe le plus proche (linéaire, polynomiale, etc.).
- Sélectionnez "Afficher l'équation sur le graphique" et vous aller voir l'équation de la courbe de tendance sur le graphique.

Instructions :

- Utiliser le programme `SauvegardeCSV.c` pour qu'il sauvegarde à intervalle régulier (2 secondes) la température de la lampe de 12 Volts pour une durée de 5 minutes avec une commande PWM constante de 50%. On aura 4 colonnes comme figure 2.1. Le numro de poste correspond au petit numéro affiché sous votre oscilloscope. Pour trouver l'adresse MAC, utiliser la commande `ifconfig` et prendre celle du `eth0`.
- ↬ Remettre le fichier CSV sous le nom **EquipeX_Temp_50_5min.csv** avec X la lettre de votre équipe.
- Ouvrir le fichier **EquipeX_Temp_50_5min.csv** avec Microsoft Excel et trouver l'équation de la varation de la température avec le temps.
- **Montrez le fichier CSV obtenu et l'équation au professeur.**
- ↬ Remettre la figure de la courbe de 5 minutes avec l'équation dans le rapport.

2.1.4 Variation de PID en fichier CSV

Dans cette partie, nous allons sauvegarder à intervalle régulier la température de la lampe de 12 Volts pendant une durée de 15 minutes avec une commande PID. Nous fixerons les valeurs des gains et nous définirons une valeure spécifique de consigne.

Instructions :

- Utiliser le programme `SauvegardeCSV.c` une autre fois pour qu'il sauvegarde à intervalle régulier (2 secondes) la température de la lampe de 12 Volts pour une durée de 15 minutes avec une commande PID. On fixe les valeurs des gains en $K_P = 10, K_I = 10, K_D = 0.2$ la consigne sera la température initiale plus 3 degrés (exemple si votre température initiale est 28.25 degrés mettre la consigne à 31 degrés). Autre que les 4 colonnes de la figure 2.1 on ajoutera 2 colonnes : **Consigne** et **Commande en %**.
- ↬ Remettre le fichier CSV sous le nom **EquipeX_Temp_PID_15min.csv** avec X la lettre

de votre équipe.

- Ouvrir le fichier **EquipeX_Temp_PID_15min.csv** avec Microsoft Excel et tracer dans la même figure la variation de la température et de la commande avec le temps.
- **Montrez les courbes au professeur.**
- ↪ Remettre la figure des deux courbes de 15 minutes. Analyser les courbes tracées et interprétez les résultats obtenus.

2.2 Partie 2 - Affichage dans une fenêtre

Dans cette partie, notre objectif est de créer une interface graphique de base qui affichera la température dans une fenêtre sur l'écran. Cette fenêtre affichera les données en temps réel. Nous utiliserons la bibliothèque GTK+ pour créer cette interface graphique. Une fois que l'interface sera créée, nous pourrons facilement acceder à la température en temps réel.

2.2.1 La bibliothèque GTK+

Dans cette partie, on va utiliser GTK+ (GIMP Toolkit) qui est une bibliothèque de fenêtrage pour le développement d'applications graphiques en C. GTK+ est conçu pour être flexible, rapide et facile à utiliser. Il fournit une variété de widgets pour la construction d'interfaces graphiques, tels que des boîtes, des boutons, des entrées de texte et des étiquettes. La bibliothèque GTK+ est largement utilisée pour la création de logiciels tels que GIMP, GNOME, et d'autres logiciels open-source.

2.2.2 Exemple de fenêtre en C

Voici dans le listing 2.2, un code C qui utilise GTK+ et affiche l'heure courante dans une fenêtre, en actualisant toutes les secondes. Le code commence par inclure les fichiers d'en-tête pour la bibliothèque GTK+. Aussi, la bibliothèque time.h est utilisée pour obtenir l'heure courante. La fonction `update_time()` est une fonction de rappel qui est appelée toutes les 2 secondes à l'aide de la fonction `g_timeout_add()`. Cette fonction récupère l'heure courante et la formate en une chaîne. La chaîne d'heure formatée est ensuite définie comme étant le texte du widget label. Dans la fonction principale, la bibliothèque GTK+ est initialisée à l'aide de la fonction `gtk_init()`. Une nouvelle fenêtre est créée à l'aide de la fonction `gtk_window_new()` et son titre est défini à l'aide de `gtk_window_set_title()`.

La largeur de la bordure de la fenêtre est définie à l'aide de `gtk_container_set_border_width()` et la taille de la fenêtre est définie à l'aide de `gtk_widget_set_size_request()`. Un nouveau widget label est créé à l'aide de la fonction `gtk_label_new()` et sa justification est définie au centre à l'aide de `gtk_label_set_justify()`. Le label est ensuite ajouté à la fenêtre à l'aide de `gtk_container_add()`.

La fonction `update_time()` est définie pour s'exécuter toutes les 2 secondes à l'aide de la fonction `g_timeout_add()`. Toute la fenêtre est ensuite affichée à l'aide de `gtk_widget_show_all()`. La fonction `g_signal_connect()` est utilisée pour connecter l'événement "destroy" de la fenêtre à la fonction `gtk_main_quit()`, qui met fin à la boucle principale GTK+ et au programme. Enfin, la boucle principale GTK+ est lancée à l'aide de la fonction `gtk_main()` et le programme se termine lorsque l'utilisateur ferme la fenêtre.

Listing 2.2 – Afficher l'heure dans une fenêtre

```c
/*
 * FenetreHeure.c
 *
 * Exemple de programme pour Afficher l'heure dans une fenêtre
 *
 *
 */
#include <gtk/gtk.h>
#include <time.h>

static gboolean update_time(gpointer label) {
time_t now = time(0);
char time_str[100];
strftime(time_str, 100, "%H:%M:%S", localtime(&now));
gtk_label_set_text(GTK_LABEL(label), time_str);
return TRUE;
}

static void destroy(GtkWidget *widget, gpointer data) {
gtk_main_quit();
}


int main(int argc, char *argv[]) {
gtk_init(&argc, &argv);

GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title(GTK_WINDOW(window), "Heure");
gtk_container_set_border_width(GTK_CONTAINER(window), 10);
gtk_widget_set_size_request(window, 100, 75);

GtkWidget *label = gtk_label_new("----");
gtk_label_set_justify(GTK_LABEL(label), GTK_JUSTIFY_CENTER);
gtk_container_add(GTK_CONTAINER(window), label);


g_timeout_add(1000, update_time, label);


gtk_widget_show_all(window);
g_signal_connect(G_OBJECT(window), "destroy", G_CALLBACK(destroy), NULL);

gtk_main();

return 0;
}
```

Instructions :

- Utiliser le programme `FenetreHeure.c` pour qu'il affiche la température de la lampe de 12 Volts toutes les secondes
- **Montrez le fonctionnement au professeur.**
- Garder ce programme, car vous allez l'utiliser dans le reste du laboratoire.

2.3 Partie 3 - Mosquitto et le MQTT

2.3.1 WiFi et Raspberry Pi

Nous travaillerons dans cette partie dans la couche réseau, au niveau du WiFi pour la communication avec l'extérieur (du Raspberry Pi !). Comme tout noeud sur le WiFi, le Raspberry Pi possède un identifiant qu'il acquiert lorsqu'il se connecte, l'adresse IP. Comme on travaille dans le standard IPv4, l'adresse IP est constitué d'une suite de 4 nombres (de type char 8 bits) séparés par des points ".".

Pour ce laboratoire, un Raspberry Pi contenant un serveur de réseau LoRaWAN est utilisé comme routeur de réseau Wifi. Ce réseau est identifié **A3564_LoRaWAN_AP**. et le mot de passe est **GSYS_3564**.

On peut obtenir cette adresse de diverses façons :

- En positionnant le curseur sur le symbole de la connexion WiFi (en haut à droite de l'écran – Figure 2.2) ;
- En utilisant la commande "ifconfig" dans le terminal de commande (Figure 2.3).

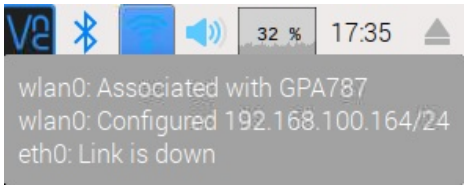


FIGURE 2.2 – Adresse du Raspberry Pi via l'icône du WiFi

```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether b8:27:eb:d4:4c:a0  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Boucle locale)
        RX packets 135726  bytes 13436668 (12.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 135726  bytes 13436668 (12.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.164  netmask 255.255.255.0  broadcast 192.168.100.255
        inet6 fe80::bfd7:e651:e0f4:8443  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:81:19:f5  txqueuelen 1000  (Ethernet)
        RX packets 63234  bytes 8979638 (8.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 71677  bytes 40542734 (38.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

FIGURE 2.3 – Résultat de la commande "**ifconfig**"

Cette adresse IP va nous servir dès que nous utilisons un équipement qui devra accéder à un serveur ou au broker MQTT du Raspberry Pi. Toutefois, si vous avez un logiciel qui communique à un serveur et que tous les deux sont dans le même Raspberry Pi, alors le vocable "localhost" est utilisé et signifie simplement d'utiliser l'adresse IP du Raspberry Pi (dans ce cas, il n'est pas nécessaire de la connaitre). Rappelons que le "localhost" est en 127.0.0.1 (section "lo" en Figure 2.3).

Toujours vérifier l'adresse IP de votre Raspberry Pi au début de la période de laboratoire, car elle peut changer d'une période de laboratoire à l'autre. L'allocation de l'adresse IP est donnée par le routeur auquel se connecte le Raspberry Pi, cette adresse peut être appelée à changer.

2.3.2 Mosquitto et le MQTT

Dans cette première partie, nous expérimenterons le protocole MQTT avec le broker Mosquitto installé dans le Raspberry Pi. Comme le protocole MQTT repose sur le TCP et l'IP, il utilise la même façon d'identifier les adresses des différents noeuds du réseau. L'adresse IP est utilisée pour identifier le broker et ses clients. Comme plusieurs liens TCP (et/ou UDP) ou autres peuvent être en fonction dans le Raspberry Pi, on ajoute à l'adresse IP le port de communication.

Le "port" est un nombre sur 2 octets (16 bits) permettant de définir qui est la source du message et à qui il est destiné. Par exemple, si un Raspberry Pi situé à l'adresse IP 127.0.0.1 (le localhost) contient un broker MQTT et est en même temps un serveur UDP, il faut être capable de savoir à qui doit être acheminé un message entrant provenant d'un client. En effet, tout client qui désire communiquer avec le Raspberry Pi doit utiliser son adresse IP.

Si un client UDP désire envoyer un message au serveur UDP situé sur le Raspberry Pi, il va indiquer qu'il désire utiliser le port 7891. Le client MQTT utilisera pour sa part le port 1883. Le MQTT sécurisé avec SLL utilise le port 8883.

Nous utiliserons ici le port 1883 puisque nous ne ferons pas du MQTT sécurisé (dans ce projet de session, les données transmises ne sont pas sensibles ou top secret...).

Installation de Mosquitto

Pour pouvoir utiliser le protocole MQTT, il faut installer Mosquitto si ce n'est déjà fait. Pour ce faire, il suffit de suivre la séquence suivante :

- `sudo apt-get update`
- `sudo apt-get install mosquitto`
- `sudo apt-get install mosquitto-clients`
- `sudo apt-get install libmosquitto-dev`

Une fois cela fait, Mosquitto est installé et démarre. A partir de maintenant, il démarrera lors de la mise en marche du Raspberry Pi. La librairie "mosquitto" utilisée en C est aussi installée par la même occasion.

2.3.3 Démarrage de Mosquitto (broker MQTT)

Généralement, une fois Mosquitto installé, le broker devrait démarrer automatiquement lors du démarrage du Raspberry Pi.

La commande "mosquitto" lance l'exécution du serveur, s'il n'est pas déjà en marche. Lorsque le serveur est déjà en marche, un message nous indique que le port est déjà utilisé (Voir figure 2.4).

```
pi@raspberrypi:~ $ mosquitto
1605286544: mosquitto version 1.4.10 (build date Wed, 13 Feb 2019 00:45:38 +0000
) starting
1605286544: Using default config.
1605286544: Opening ipv4 listen socket on port 1883.
1605286544: Error: Address already in use
```

FIGURE 2.4 – Lancement de Mosquitto

On peut voir si le broker mosquitto est en marche à l'aide de la commande "`netstat -ta`". On peut voir le résultat à la Figure 2.5 ou le port 1883 est affiché dans la liste des processus TCP. Vous pouvez voir par la même occasion tous les ports des processus du TCP dans le Raspberry Pi.

```
pi@raspberrypi:~ $ netstat -ta
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale          Adresse distante        Etat
tcp        0      0 0.0.0.0:5900            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5901            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:x11-1           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1880            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1883            0.0.0.0:*               LISTEN
tcp        0      0 localhost:50098         localhost:36713         ESTABLISHED
tcp        0      0 localhost:36838         localhost:32983         ESTABLISHED
tcp        0      0 localhost:36713         localhost:50098         ESTABLISHED
tcp        0      0 localhost:32983         localhost:36838         ESTABLISHED
tcp        0      0 raspberrypi:5901        GSYS050716:63239        ESTABLISHED
tcp6       0      0 [::]:5900               [::]:*                  LISTEN
tcp6       0      0 [::]:5901               [::]:*                  LISTEN
tcp6       0      0 [::]:x11-1              [::]:*                  LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
tcp6       0      0 [::]:1883               [::]:*                  LISTEN
```

FIGURE 2.5 – Utilisation de la commande netstat -ta

2.3.4 Publieurs et abonnés

Nous allons maintenant faire des tests à partir du terminal. Il faudra ouvrir deux fenêtres de terminal. L'une servira à publier et l'autre servira à s'abonner. Vous trouverez les détails des commandes "`mosquitto_sub`" et "`mosquitto_pub`" dans les notes de cours (Chapitre 10).

Commençons par des commandes minimalistes. Dans l'un des terminaux, écrire la commande

```
mosquitto_sub -h localhost -t donnee
```

Cela fait en sorte que le broker situé dans le Raspberry Pi (**localhost**) possède un abonné au topic "**donnee**". Vous constaterez que le terminal attend les publications du publieur sur le topic "**donnee**". Dans l'autre terminal, écrire la commande

```
mosquitto_pub -h localhost -t donnee -m "Bonjour␣tout␣le␣monde"
```

Si la chaine de caractères comporte des espaces, il faut la mettre entre guillemets ("). Vous constaterez que le message apparait dans le terminal de l'abonné. Répétez avec commande

```
mosquitto_pub -h localhost -t donnee -m 22.5
```

On peut ajouter des options supplémentaires en ajoutant des "flags" à ces commandes. Ainsi,

```
mosquitto_pub -h localhost -t donnee m 22.5 -d
```

fait en sorte que le détail de la transaction est mentionné (voir Figure 2.6). On constate que le publieur envoie un signal de connexion (CONNECT), reçoit de la part du broker une confirmation de connexion (CONNACK), publie le message (PUBLISH) et se déconnecte (DISCONNECT). Le flag "**-d**" aide au diagnostic. Vous pouvez le faire aussi du côté du client. La Figure 2.7 montre un exemple de transaction lorsque l'option "**-d**" est active pour l'abonné.

```
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t donnee -m 22.5 -d
Client mosqpub|7545-raspberryp sending CONNECT
Client mosqpub|7545-raspberryp received CONNACK (0)
Client mosqpub|7545-raspberryp sending PUBLISH (d0, q0, r0, m1, 'donnee', ...
 (4 bytes))
Client mosqpub|7545-raspberryp sending DISCONNECT
pi@raspberrypi:~ $
```

FIGURE 2.6 – Exemple de publication avec mode diagnostic (debug) activé "flag" "-d"

```
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t sujet -m 22.5 -d
Client mosqpub/4763-raspberryp sending CONNECT
Client mosqpub/4763-raspberryp received CONNACK
Client mosqpub/4763-raspberryp sending PUBLISH (d0, q0, r0, m1, 'sujet', ... (4
bytes))
Client mosqpub/4763-raspberryp sending DISCONNECT
pi@raspberrypi:~ $ scrot
^C
pi@raspberrypi:~ $ mosquitto_sub -h localhost -t sujet -d
Client mosqsub/4762-raspberryp sending CONNECT
Client mosqsub/4762-raspberryp received CONNACK
Client mosqsub/4762-raspberryp sending SUBSCRIBE (Mid: 1, Topic: sujet, QoS: 0)
Client mosqsub/4762-raspberryp received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/4762-raspberryp received PUBLISH (d0, q0, r0, m0, 'sujet', ... (4 bytes))
22.5
```

FIGURE 2.7 – Exemple de transaction MQTT avec mode diagnostic activé pour les deux clients.

On peut modifier la qualité du service (QoS) en insérant l'option "**-q** x" avec x égal à 0, 1, ou 2. Si on ne le spécifie pas, la qualité du service est (par défaut) de 0. Faire un test en activant une qualité de service de 2 pour le publieur

```
mosquitto_pub -h localhost -t donnee -q 2 -m 33.3 -d
```

et en conservant l'option de diagnostic. Vous constaterez dans le terminal du publieur (Figure 2.8) qu'une transaction avec une qualité de service de 2 implique plus d'échanges entre le broker et le publieur. Si on n'a pas changé la qualité de service du côté client, seul un PUBLISH est fait par le broker, car la qualité de service est de 0. Si vous changez la qualité de service du côté client, en la mettant à 1 par exemple, la transaction devient lors plus complexe (Figure 2.9).

```
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t sujet -q 2 -m 33.3 -d
Client mosqpub/4787-raspberryp sending CONNECT
Client mosqpub/4787-raspberryp received CONNACK                    : sujet, QoS: 0)
Client mosqpub/4787-raspberryp sending PUBLISH (d0, q2, r0, m1, 'sujet', ... (4
bytes))
Client mosqpub/4787-raspberryp received PUBREC (Mid: 1)            , 'sujet', ... (4 bytes))
Client mosqpub/4787-raspberryp sending PUBREL (Mid: 1)
Client mosqpub/4787-raspberryp received PUBCOMP (Mid: 1)
Client mosqpub/4787-raspberryp sending DISCONNECT
pi@raspberrypi:~ $ scrot
Client mosqsub/4762-raspberryp received PINGRESP
Client mosqsub/4762-raspberryp sending PINGREQ
Client mosqsub/4762-raspberryp received PINGRESP
Client mosqsub/4762-raspberryp sending PINGREQ
Client mosqsub/4762-raspberryp received PINGRESP
Client mosqsub/4762-raspberryp sending PINGREQ
Client mosqsub/4762-raspberryp received PINGRESP
Client mosqsub/4762-raspberryp received PUBLISH (d0, q0, r0, m0, 'sujet', ... (4 bytes))
33.3
```

FIGURE 2.8 – Transaction avec publieur ayant une QoS de 2 et un abonné ayant une QoS de 0

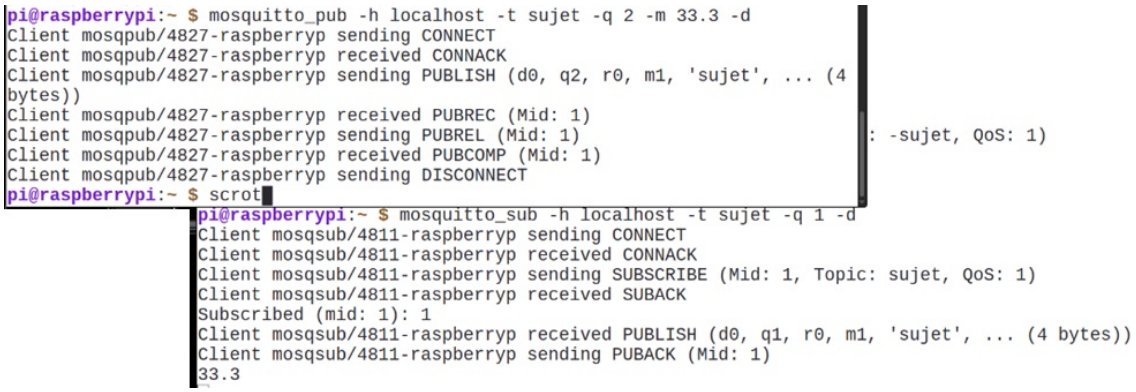


FIGURE 2.9 – Transaction avec publieur ayant une QoS de 2 et un abonné ayant une QoS de 1

La séquence PINGREQ et PINGRESP apparaissant dans les transactions du côté client (Figure 2.9) provient du fait que ce dernier doit manifester son existence à intervalle régulier au broker. Si le client quitte de façon normale avec un DISCONNECT, le broker considère que la fin de la transaction est normale. Si le broker ne reçoit plus de PINGREQ depuis un certain laps de temps prédéterminé et qu'il n'a pas reçu le signal DISCONNECT, cela signifie que le client a quitté abruptement et le broker publiera les dernières volontés du client, si celui-ci en avait fait part au broker.

Si on utilise l'option `{-v}` (verbose) pour l'abonné, on retrouve en plus du message le topic de celui-ci. On peut même envoyer un fichier via le MQTT, avec l'option `-f}`, suivit du nom du fichier à transmettre.

Prenez le temps d'essayer quelques commandes de publication et d'abonnement pour voir comment cela marche [1]. Essayez différents niveaux de qualité de service pour le publieur et l'abonné (il y a 9 combinaisons possibles) et observez le résultat.

2.3.5 MQTT dans un programme en C

Avant de se lancer dans le coeur du projet, prenons le temps de s'initier à comment intégrer les appels MQTT dans un programme en C [2]. Cela a été abordé brièvement au Chapitre 10 des notes de cours. Les programmes mentionnés ici seront dans un fichier ".zip" disponible sur le site Moodle du cours.

Publieur minimaliste

Le fichier `publieur_minimaliste_MQTT.c` contient le code pour un publieur dont le seul topic qu'il publie est "donnee" et dont la qualité de service est de 0. La publication n'a lieu qu'une seule fois (voir les lignes 39 et 40 du Listing 2.3). La Figure 2.10 montre le résultat du côté de l'abonné (sur le terminal). Même si l'abonné est à une qualité de service de 1 `mosquitto_sub -h localhost -t donnee -q 1 -d` la transaction apparait avec une qualité de service de 0, car le message initial fait par le publieur était de qualité de service égale à 0.

1. Le lien suivant vous mène vers la description de la commande `mosquitto_pub` : `https://mosquitto.org/man/mosquitto_pub-1.html`. Cet autre lien vous mène vers la description de la commande `mosquitto_sub` : `https://mosquitto.org/man/mosquitto_sub-1.html`.
2. Le lien suivant vous mène vers la description des fonctions de la librairie "mosquitto.h" : `https://mosquitto.org/api/files/mosquitto-h.html`.

Listing 2.3 – Code C générant la publication en QoS 0

```
1 sprintf(text , "%5.2f?C" ,23.75);
2 ret = mosquitto_publish (mosq , NULL , MQTT_TOPIC1 , strlen(text), text , 0,
      false);
```

```
pi@raspberrypi:~ $ mosquitto_sub -h localhost -t donnee -q 1 -d
Client mosqsub|7647-raspberryp sending CONNECT
Client mosqsub|7647-raspberryp received CONNACK (0)
Client mosqsub|7647-raspberryp sending SUBSCRIBE (Mid: 1, Topic: donnee, QoS:
 1)
Client mosqsub|7647-raspberryp received SUBACK
Subscribed (mid: 1): 1
Client mosqsub|7647-raspberryp received PUBLISH (d0, q0, r0, m0, 'donnee', ..
. (7 bytes))
23.75 C
```

FIGURE 2.10 – Résultat du côté abonné du publieur minimaliste

Publieur plus élaboré

Le fichier `publieur_MQTT.c` contient le code pour un publieur légèrement plus élaboré publiant deux topics, "capteur/zone1/temperature" et "capteur/zone1/pression" (Lignes 40 à 50 du Listing 2.4). La qualité de service du premier topic est de 0, tandis que celui du second est 1. La publication n'a lieu qu'une fois pour chaque sujet. La Figure 2.11 montre le résultat du côté de l'abonné (sur le terminal) lorsqu'il est abonné aux deux topics avec une qualité de service de 1 :

```
mosquitto_sub -h localhost -t capteur/zone1/# -q 1 -d -v
```

La transaction du premier topic apparait avec une qualité de service de 0, car le message initial fait par le publieur était de qualité de service égale à 0. Le second topic étant de qualité de service 1 du côté publieur, il le sera aussi du côté abonné qui est configuré à cette qualité de service (d'où le PUBACK - confirmation de publication - apparaissant dans cette transaction).

Listing 2.4 – Code C du publieur avec deux publications successives de deux QoS différentes

```
1         sprintf(text, "%5.2f␣C",23.75);
2         ret = mosquitto_publish(mosq, NULL, MQTT_TOPIC1, strlen(text), text,
              0, false);
3         if (ret)
4         {
5                 fprintf(stderr,"Ne␣peut␣publier␣sur␣le␣serveur␣Mosquitto\n")
                      ;
6                 exit(-1);
7         }
8         sleep(1);
9         sprintf(text, "%8.2f␣Pa",101325.2);
10        ret = mosquitto_publish(mosq, NULL, MQTT_TOPIC2, strlen(text), text,
              1, false);
```

```
pi@raspberrypi:~ $ mosquitto_sub -h localhost -t capteur/zone1/# -q 1 -d -v
Client mosqsub/5125-raspberryp sending CONNECT
Client mosqsub/5125-raspberryp received CONNACK
Client mosqsub/5125-raspberryp sending SUBSCRIBE (Mid: 1, Topic: capteur/zone1/#, QoS: 1)
Client mosqsub/5125-raspberryp received SUBACK
Subscribed (mid: 1): 1
Client mosqsub/5125-raspberryp received PUBLISH (d0, q0, r0, m0, 'capteur/zone1/temperature',
.. (7 bytes))
capteur/zone1/temperature 23.75 C
Client mosqsub/5125-raspberryp received PUBLISH (d0, q1, r0, m1, 'capteur/zone1/pression', ...
(12 bytes))
Client mosqsub/5125-raspberryp sending PUBACK (Mid: 1)
capteur/zone1/pression 101325.20 Pa
```

FIGURE 2.11 – Résultat de l'exécution du côté client qui est en QoS de 1

Abonné minimaliste

Le fichier `abonne_minimaliste_MQTT.c` contient le code pour un abonné qui suit les publications du topic "donnee" et dont la qualité de service est de 1. La lecture des messages entrants se fait pendant 120 secondes, ensuite, le programme s'arrête.

La Figure 2.12 montre le résultat du côté de l'abonné aux publications faites en Figure 2.13, exemple : `mosquitto_pub -h localhost -t donnee -q 1 -m GPA` les transactions étant toutes de qualité de service égale à 1.

```
Topic : sujet  --> Message : 33.3
Topic : sujet  --> Message : Hello
Topic : sujet  --> Message : Moustique
Topic : sujet  --> Message : GPA


------------------
(program exited with code: 0)
Press return to continue
```

FIGURE 2.12 – Résultat de l'exécution du programme d'abonné minimaliste dans le terminal de Geany

```
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t donnee -q 1 -m 33.3
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t donnee -q 1 -m 33.3
pi@raspberrypi:~ $
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t donnee -q 1 -m GPA
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t donnee -q 1 -m "Bonjour tout
 le monde"
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t donnee -q 1 -m 3.1416
pi@raspberrypi:~ $
```

FIGURE 2.13 – Publication faites sur le topic "donnee"

Abonné un peu plus élaboré

Le fichier abonne_MQTT.c contient le code pour un abonné qui suit les publications concernant la température dans diverses zones (topic : capteur/+/temperature") et la pression atmosphérique (topic : \lstinlinecapteur/zone1/pression") dont la qualité de service est de 1. La lecture des messages entrants se fait pendant 120 secondes, ensuite, le programme s'arrête. La Figure 2.14 montre le résultat du côté de l'abonné et la Figure 2.15 montre les publications correspondantes, exemple :

```
mosquitto_pub -h localhost -t capteur/zone3/temperature -q 1 -m "24.4 C"-d
```

les transactions étant toutes de qualité de service égale à 1.

```
Topic : capteur/zone1/temperature  --> Message : 22.1 C
La temperature dans le salon est de 22.100000 C.
Topic : capteur/zone2/temperature  --> Message : 23.6 C
La temperature dans la chambre est de 23.600000 C
Topic : capteur/zone3/temperature  --> Message : 24.0 C
La temperature dans la cuisine est de 24.000000 C
Topic : capteur/zone1/pression  --> Message : 101150 Pa
La pression atmospherique est de 101150.000000 Pa.
```

FIGURE 2.14 – Résultat du terminal pour le code abonne_MQTT.c

```
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t capteur/zone1/temperature -q 1
-m "22.1 C" -d
Client mosqpub/5388-raspberryp sending CONNECT
Client mosqpub/5388-raspberryp received CONNACK
Client mosqpub/5388-raspberryp sending PUBLISH (d0, q1, r0, m1, 'capteur/zone1/t
emperature', ... (6 bytes))
Client mosqpub/5388-raspberryp received PUBACK (Mid: 1)
Client mosqpub/5388-raspberryp sending DISCONNECT
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t capteur/zone2/temperature -q 1
-m "23.6 C" -d
Client mosqpub/5389-raspberryp sending CONNECT
Client mosqpub/5389-raspberryp received CONNACK
Client mosqpub/5389-raspberryp sending PUBLISH (d0, q1, r0, m1, 'capteur/zone2/t
emperature', ... (6 bytes))
Client mosqpub/5389-raspberryp received PUBACK (Mid: 1)
Client mosqpub/5389-raspberryp sending DISCONNECT
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t capteur/zone3/temperature -q 1
-m "24.0 C" -d
Client mosqpub/5390-raspberryp sending CONNECT
Client mosqpub/5390-raspberryp received CONNACK
Client mosqpub/5390-raspberryp sending PUBLISH (d0, q1, r0, m1, 'capteur/zone3/t
emperature', ... (6 bytes))
Client mosqpub/5390-raspberryp received PUBACK (Mid: 1)
Client mosqpub/5390-raspberryp sending DISCONNECT
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t capteur/zone1/pression -q 1 -m
```

FIGURE 2.15 – Quelques-unes des publications envoyées à l'abonné

Abonné Publieur

Le fichier `abonne_publieur_MQTT.c` contient le code pour un abonné au sujet `sujet` qui republie le message reçu sous le sujet `copie`. On peut voir dans les Figures 2.16, 2.17 et 2.18 les différentes captures d'écran.

```
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t sujet -q 1 -m GPA -d
Client mosqpub/5499-raspberryp sending CONNECT
Client mosqpub/5499-raspberryp received CONNACK
Client mosqpub/5499-raspberryp sending PUBLISH (d0, q1, r0, m1, 'sujet', ... (6
bytes))
Client mosqpub/5499-raspberryp received PUBACK (Mid: 1)
Client mosqpub/5499-raspberryp sending DISCONNECT
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t sujet -q 1 -m "Le temps passe"
-d
Client mosqpub/5503-raspberryp sending CONNECT
Client mosqpub/5503-raspberryp received CONNACK
Client mosqpub/5503-raspberryp sending PUBLISH (d0, q1, r0, m1, 'sujet', ... (14
 bytes))
Client mosqpub/5503-raspberryp received PUBACK (Mid: 1)
Client mosqpub/5503-raspberryp sending DISCONNECT
pi@raspberrypi:~ $ mosquitto_pub -h localhost -t sujet -q 1 -m "104555 Pa" -d
Client mosqpub/5504-raspberryp sending CONNECT
Client mosqpub/5504-raspberryp received CONNACK
Client mosqpub/5504-raspberryp sending PUBLISH (d0, q1, r0, m1, 'sujet', ... (9
bytes))
Client mosqpub/5504-raspberryp received PUBACK (Mid: 1)
Client mosqpub/5504-raspberryp sending DISCONNECT
```

FIGURE 2.16 – Publications qui seront traitées par `abonne_publieur_MQTT`.

```
Fichier  Édition  Onglets  Aide
Topic : sujet  --> Message :  GPA
Topic : sujet  --> Message : Le temps passe
Topic : sujet  --> Message : 104555 Pa
```

FIGURE 2.17 – Le terminal du programme et les messages reçus

```
pi@raspberrypi:~ $ mosquitto_sub -h localhost -t sujet -q 2 -v
sujet GPA
sujet Le temps passe
sujet 104555 Pa
```

FIGURE 2.18 – L'abonné recevant les publications du programme `abonne_publieur_MQTT.c`

Vous pouvez tester votre programme avec le terminal via `mosquitto_pub` et `mosquitto_sub`. Vous le testerez aussi avec votre téléphone cellulaire, via des applications MQTT (par exemple : MQTT Dash dans Android ou MQTTool dans iOS).

Instructions :

- Modifier le programme `publieur_MQTT.c` pour qu'il publie à intervalle régulier (2 secondes) la température de la lampe de 12 Volts (donc, insérer l'appel au SPI et la librairie bcm2835). Le programme devra s'arrêter suit à l'appui sur le bouton rouge.
- Dans un autre listing, modifier le programme `abonne_MQTT.c` pour qu'il affiche à intervalle régulier la température de la lampe de 12 Volts mesurée par le programme que vous avez fait précédemment pour publier cette information. Le programme devra s'arrêter suit à l'appui sur le bouton rouge. Les deux programmes devront donc fonctionner simultanément.
- Afficher la valeur reçue par l'abonné dans une fenêtre (avec GTK+).
- **Montrez le fonctionnement au professeur.**
- Faire un nouveau programme utilisant la commande PID faite au laboratoire 1 pour en faire un publieur de la température du thermocouple, la consigne de température, le pourcentage PWM calculé, et de l'état désiré des deux DEL (QoS de 2) le code doit aussi fonctionner pour un abonné au 5 sujets (QoS de votre choix). Les valeurs des gains du PID sont fixées dans le programme et ne peuvent être modifiées, comme dans un thermostat électronique disponible sur le marché. On fixe ces valeurs en $K_P = 10, K_I = 10, K_D = 0.2$
- ↬ Remettre le code en C résultant.
- ↬ Remettre des captures d'écran de l'abonné et du publieur.
- Utiliser une application MQTT (par exemple : MQTT Dash dans Android ou MQTTool dans iOS) pour s'abonner la température de la lampe de 12 Volts.
- **Montrez le fonctionnement au professeur.**
- ↬ Remettre des captures d'écran du téléphone cellulaire (ou tablette) utilisant l'application MQTT.

2.4 Partie 4 - Node-Red

Dans cette partie, vous combinerez l'environnement HMI Node-Red avec le MQTT. Le programme de commande PID avec MQTT ayant été programmé dans la section précédente, il reste à introduire l'utilisation de l'interface Node-Red. L'usage de Node-Red permet de passer outre les problèmes des applications MQTT (en particulier dans l'environnement iOS). En effet, l'accès à l'interface Node-Red se fera au travers d'un navigateur WEB, ce qui facile beaucoup la portabilité sur différentes plates-formes.

2.4.1 Lancement de Node-Red

Le lancement de Node-Red se fait en tapant la commande `node-red` dans le terminal. Ensuite, on doit ouvrir un navigateur WEB pour développer et afficher l'interface Node-Red.

Si vous êtes sur le Raspberry Pi ou vous développerez votre interface, il suffit d'utiliser l'adresse IP suivante : `127.0.0.1:1880`. Si vous êtes sur un ordinateur sur le même réseau (connecté au même routeur), lancer un navigateur et utiliser l'adresse IP du Raspberry Pi suivit de `":1880`. L'interface s'ouvre alors et ressemble à ce que l'on voit en Figure 2.19.

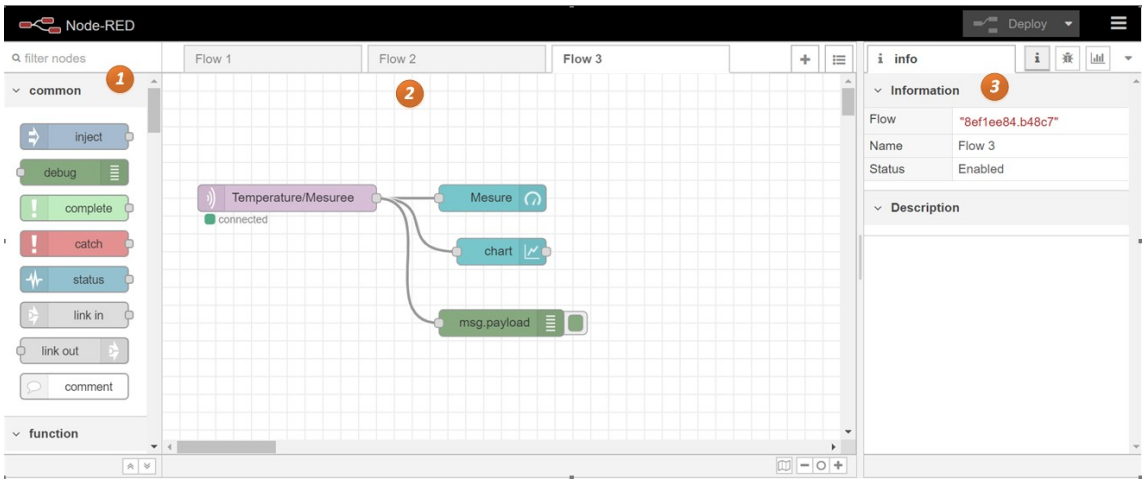


FIGURE 2.19 – Interface Node-Red

La zone 1, nommée la "Palette" contient la liste des noeuds que l'on peut utiliser pour développer le programme (via schéma bloc, que l'on nomme un "flux") de l'interface. La zone 2 contient les "flux" (en anglais flow) ou schémas blocs constituant le programme Node-Red. La zone 3, nommée "barre latérale" est utilisée pour obtenir de l'information sur les schémas blocs, faire du débogage et concevoir les interfaces HMI.

2.4.2 Premier programme Node-RED

Ce premier programme permet de prendre contact avec cette interface de programmation et il est montré en Figure 2.20.

On constate que l'on utilise deux noeuds `inject` et un noeud `debug` que l'on voit dans le menu des noeuds, dans la palette à gauche. Pour amener un noeud dans la surface de travail (à droite dans la Figure 2.20), il suffit de le cliquer et le draguer du menu vers la surface de travail.

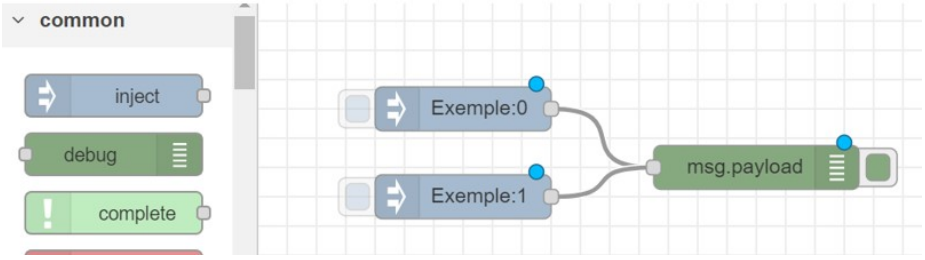


FIGURE 2.20 – Exemple de programme Node-Red

En double cliquant un des noeuds `inject` dans la surface de travail, on obtient un menu de configuration dans la zone de travail et de l'information sur la configuration du noeud dans la zone `info` à droite (Figure 2.21).

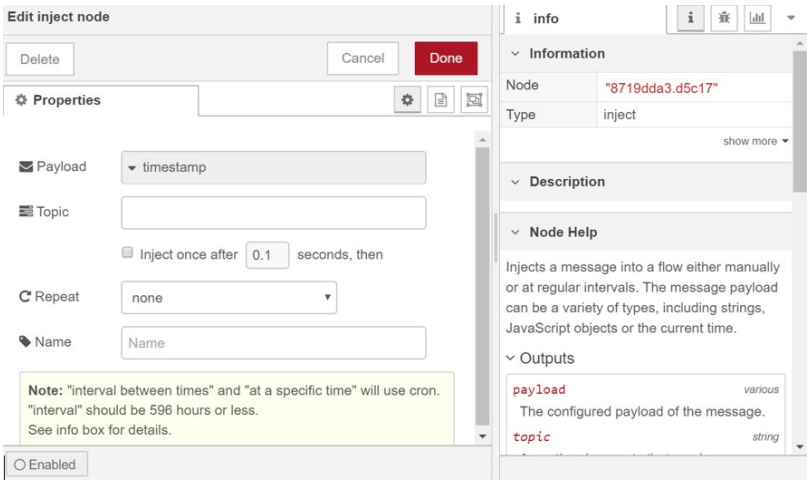


FIGURE 2.21 – Menu de configuration d'un noeud `inject`

Dans notre cas, on choisit d'entrer deux valeurs numériques (une par noeud `inject`). Pour ce faire on modifie la charge utile du noeud (payload) et on choisit l'option nombre (number – Figure 2.22). On inscrit aussi la valeur numérique (dans un cas `0`, dans l'autre `1` – Figure 2.23). On peut aussi définir le topic de ce noeud (c'est optionnel). Dans cet exemple, j'ai mis le topic `Exemple` pour les deux noeuds.

Un noeud `debug` est ajouté, il affichera la charge utile du noeud qui lui enverra un signal. On trace ensuite les liens (ou câbles) entre les noeuds en partant du petit carré gris situé à gauche du noeud `inject` et en l'amenant vers le petit carré gris situé à droite du noeud `debug`.

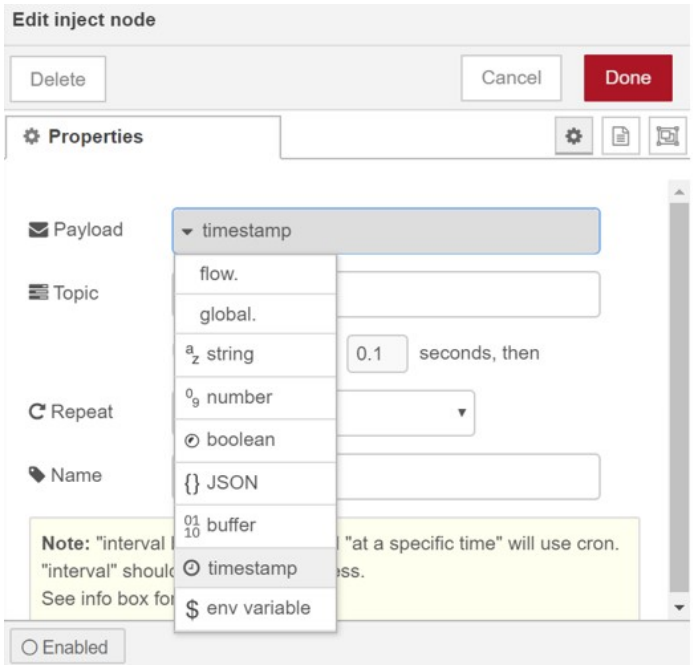


FIGURE 2.22 – Configuration du type de charge utile (payload) du message du noeud `inject`

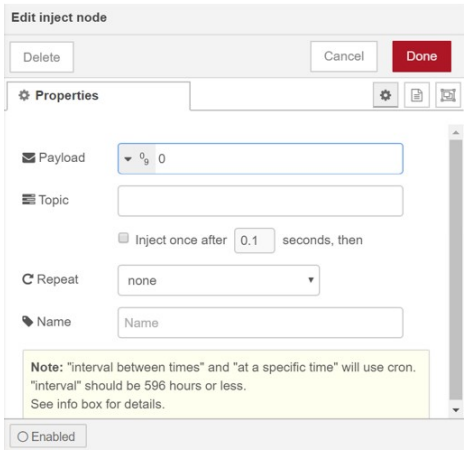


FIGURE 2.23 – Exemple avec une charge utile numérique d'un noeud `inject`

FIGURE 2.24 – Bouton `Deploy` et haut de la barre latérale

Une fois le programme terminé, on en lance l'exécution en cliquant sur le bouton `Deploy` situé dans l'entête (Figure 2.24). Pour pouvoir débugger le programme on clique sur l'icône d'insecte (Figure 2.24) situé juste en dessous du bouton `Deploy`. On teste le programme en cliquant sur les carrés gris localisés à gauche des noeuds `inject`.

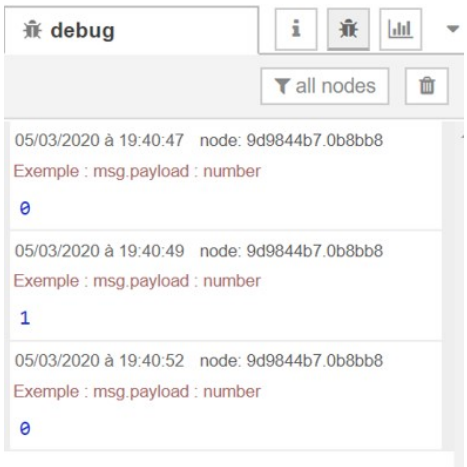


FIGURE 2.25 – Écran de débogage

On constate l'apparition de messages dans la zone de débogage. Ces messages nous indiquent le type de charge utile, le contenu de la charge utile, l'identifiant du noeud de débogage, l'heure et la date du message (Figure 2.25).

Nous allons maintenant découvrir différents noeuds et nous insisterons sur ceux qui seront utilisés dans le projet.

2.4.3 Les Noeuds de Node-RED

Noeuds en entrée

Les noeuds d'entrée sont habituellement utilisés pour récupérer les événements extérieurs. Trois d'entre eux sont montrés en Figure 2.26. On constate la sortie de ces noeuds représentés par le petit carré gris localisé à droite de ces noeuds.

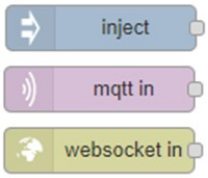


FIGURE 2.26 – Exemples de noeuds en entrée

Le noeud `inject` permet d'envoyer des valeurs préétablies dans un flux. Comme on l'a constaté dans la sous section précédemment, il est utile pour faire du débogage.

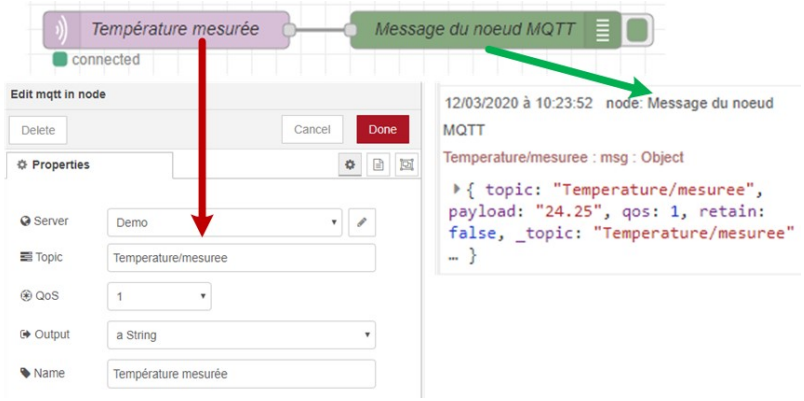


FIGURE 2.27 – Exemple d'utilisation du noeud `mqtt in`

Le noeud `mqtt in` sera utile dans ce projet pour s'abonner à des topics MQTT. La Figure 2.27 montre comment configurer ce noeud. On doit sélectionner le serveur MQTT, définir le topic auquel on s'abonne, la qualité de service (0, 1 ou 2) et le type de sortie.

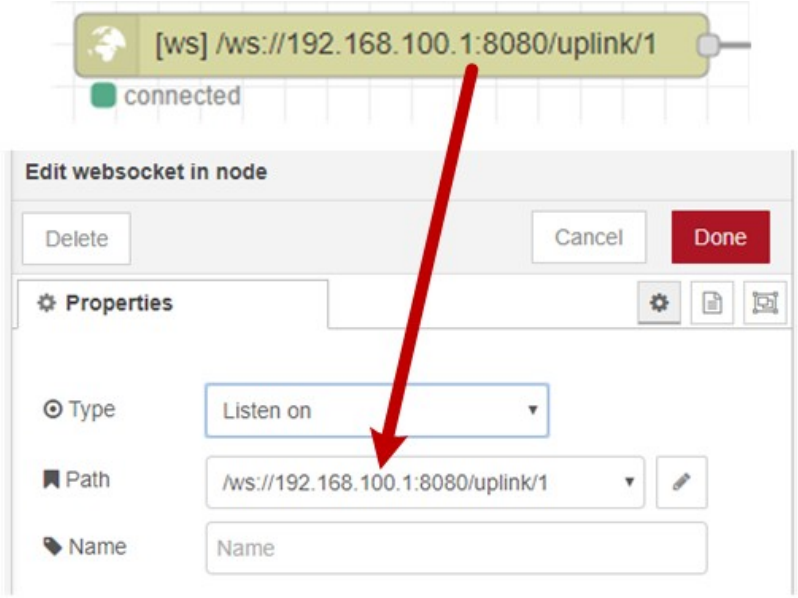


FIGURE 2.28 – Exemple d'utilisation du noeud `websocket in`

Le noeud `websocket in` permet de récupérer des websockets reçus (voir Figure 2.28). Ce noeud servira dans la partie suivante du projet pour récupérer des mesures faites par divers capteurs Bluetooth. On doit choisir le type d'interaction (ici on intercepte les données) et l'adresse du serveur de websockets.

Noeuds en sortie

Les noeuds en sortie permettent d'interagir avec l'environnement extérieur. La Figure 2.29 en montre deux. L'entrée de ces noeuds, représentée par le petit carré gris, est localisé du côté gauche du noeud.

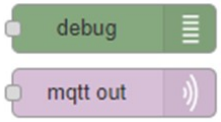


FIGURE 2.29 – Exemples de noeuds en sortie

Le noeud `debug` permet de faire le débogage du flux en observant le contenu des signaux transportés par les câbles. On peut observer le contenu de la charge utile, le contenu du message complet (charge utile, topic, identification du message, ...). On l'a testé dans un exemple précédent.

Le noeud `mqtt out` permet de publier des messages vers le broker MQTT. Dans ce projet, il servira à fournir les consignes de température et de niveau à notre réservoir virtuel. La Figure 2.30 montre un exemple de configuration d'un publieur MQTT. On doit sélectionner le serveur MQTT, définir le topic sur lequel on publie, la qualité de service (0, 1 ou 2) et indiquer si la dernière publication doit être conservée par le broker (retain).

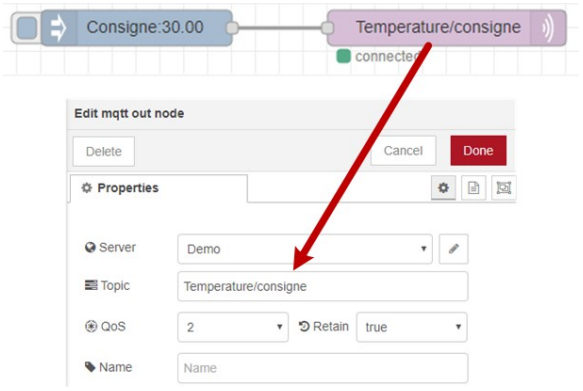


FIGURE 2.30 – Exemple d'utilisation du noeud `mqtt out`

Configuration d'un serveur MQTT

En Figure 2.30, on retrouve comme paramètre le "serveur" du service MQTT. Si aucun serveur n'existe ou que celui que vous voulez utiliser n'est pas dans la liste, il faut créer un nouveau serveur en cliquant sur le crayon à droite du menu déroulant pour choisir le serveur.

En cliquant sur le crayon, une fenêtre apparait (Figure 2.31.a) pour permettre de configurer la connexion à un serveur (broker) MQTT. On lui donne un nom, on indique son adresse (localhost si le broker est sur le même Raspberry Pi que l'environnement Node-Red). La Figure 2.31.b montre un exemple ou la connexion configurée est identifiée par "Reservoir_virtuel".

Le noeud contenant cette configuration est nommé un "noeud de configuration". Ces noeuds n'apparaissent pas sur les "flow". Pour connaitre les "noeuds de configuration" existants, on peut cliquer sur la petite flèche à droite de l'insecte (Figure 2.32.a) et sur le menu qui apparait, on clique sur "Configuration nodes". La Figure 2.32.b montre un exemple d'affichage des noeuds de configuration et on y voit le noeud créé précédemment en Figure 2.31.b.

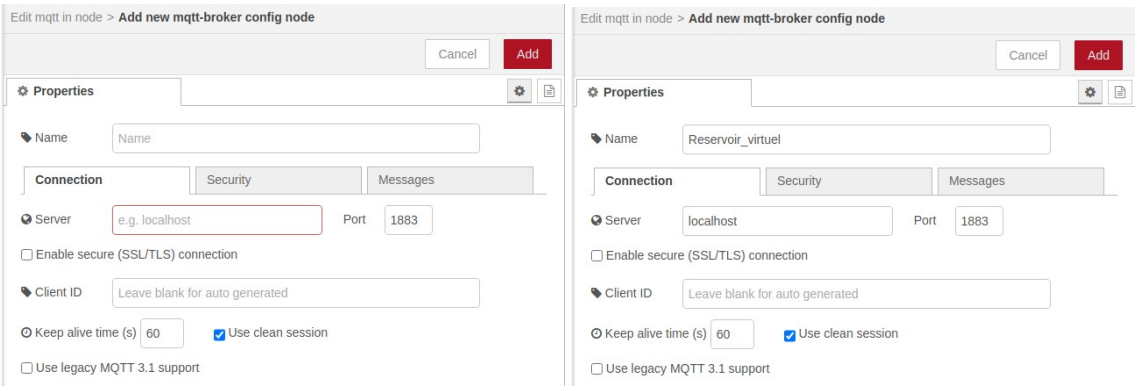
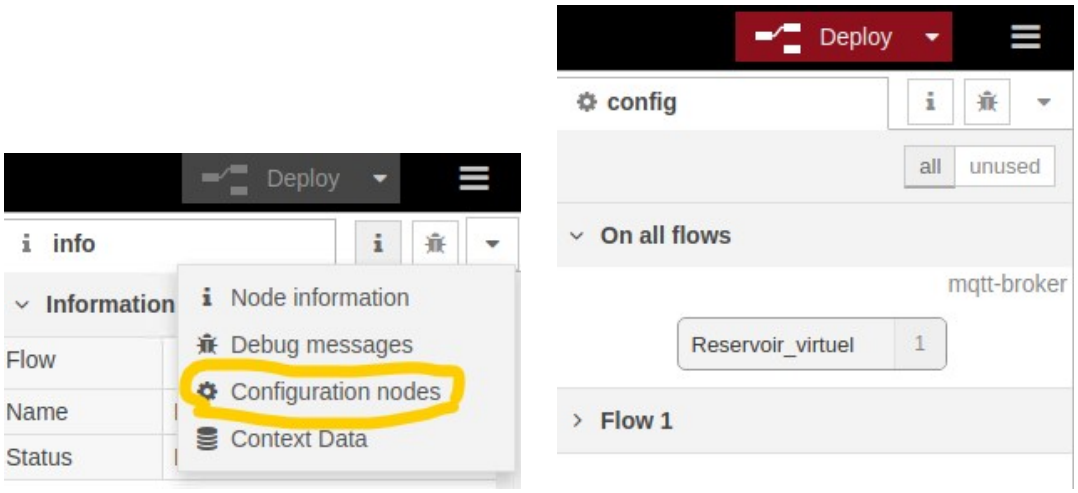


FIGURE 2.31 – (a) Fenêtre de configuration d'un nouveau "broker" MQTT (b) Configuration d'un "broker" MQTT nommé "Reservoir_virtuel"



(a) (b)

FIGURE 2.32 – Menu menant à l'affichage de la liste des noeuds de configuration

Noeuds de fonction

Les noeuds de fonction comprennent diverses fonctions préétablies et un noeud de fonction ou vous pouvez y inscrire un petit programme en JavaScript. Deux noeuds de fonction que vous utiliserez dans ce projet sont montrés en Figure 2.33. On remarque qu'un noeud de fonction comporte à la fois des entrées et des sorties (petits carrés gris).

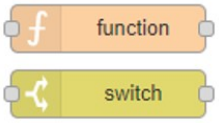


FIGURE 2.33 – Exemples de noeuds de fonction

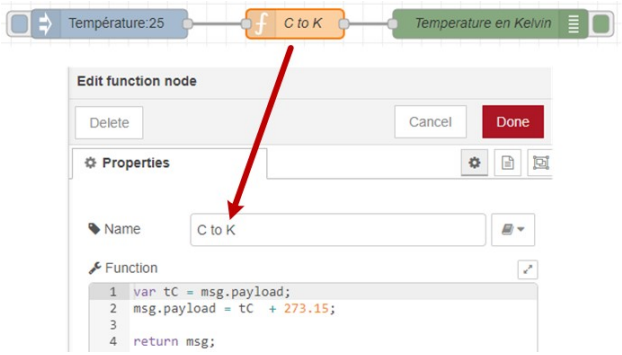


FIGURE 2.34 – Exemples de noeuds de fonction

Le noeud `function` permet d'entrer un programme en JavaScript permettant de définir une fonction personnalisée. La Figure 2.34 montre une petite fonction convertissant une température

de °C vers Kelvin. La température entrant dans la fonction (à gauche) possède une charge utile msg.payload reçue du noeud `inject` à gauche. La fonction modifie la charge utile en lui additionnant 273.15 et le message avec cette nouvelle charge utile continue vers le noeud de débogage à droite. Idéalement, on devrait nommer la fonction pour aider à identifier les différentes fonctions présentes dans les flux.

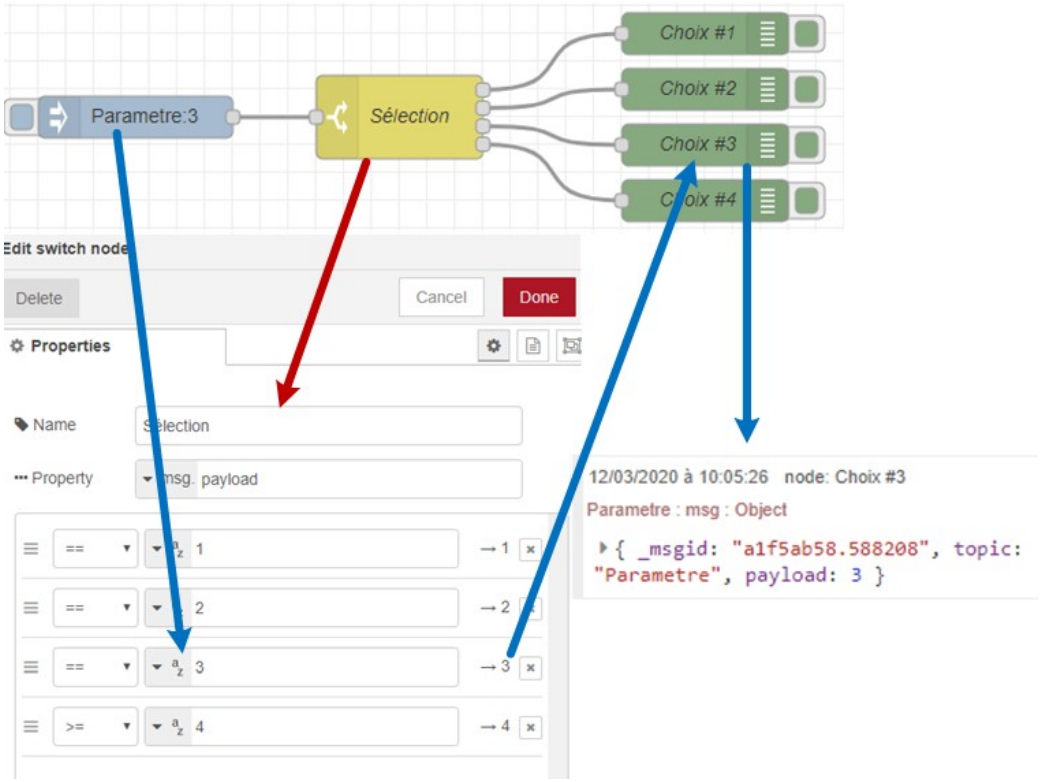


FIGURE 2.35 – Exemple d'usage du noeud `switch`

Le noeud `switch` permet d'acheminer le message vers une des sorties de ce noeud. La Figure 2.35 montre un exemple de l'usage de ce noeud. La valeur injectée par le noeud de gauche est 3, le noeud `switch` transmet alors le message vers la sortie #3 et le noeud de débogage "Choix #3" reçoit ce message qui apparait alors dans la zone de débogage. On indique le nom identifiant le noeud, la propriété à tester et la liste des tests à faire pour transmettre le message vers une sortie donnée.

Instructions :

- En utilisant un noeud `debug`, afficher la température mesurée du thermocouple, la consigne et la commande.
- **Montrez le fonctionnement au professeur.**
- ↬ Faire des captures d'écran montrant le flux et expliquer les configurations faites.
- ↬ *Optionnel - (Bonus 5pts) :* J'aimerai avoir votre opinion. Indiquez-moi ce que vous pensez de ce projet de session. Quels sont les points à améliorer ? Le cours étant en construction et en processus d'amélioration continue, votre opinion m'aidera sur ce point.